
HTTPretty Documentation

Release 1.1.1

Gabriel Falcao

May 14, 2021

CONTENTS

1	What is HTTPretty ?	3
1.1	A more technical description	3
1.2	Installing	4
2	Demo	5
2.1	expecting a simple response body	5
2.2	making assertions in a callback that generates the response body	5
2.3	Link headers	6
3	Motivation	7
4	Acknowledgements	9
4.1	Caveats	9
5	API Reference	11
5.1	register_uri	11
5.2	enable	12
5.3	disable	13
5.4	is_enabled	13
5.5	last_request	13
5.6	latest_requests	14
5.7	activate	14
5.8	httprettified	14
5.9	enabled	15
5.10	httprettized	15
5.11	HTTPrettyRequest	15
5.12	HTTPrettyRequestEmpty	16
5.13	FakeSockFile	16
5.14	FakeSSLSocket	17
5.15	URIInfo	17
5.16	URIMatcher	18
5.17	Entry	18
6	Modules	19
6.1	Core	19
6.2	Http	28
6.3	Utils	28
6.4	Exceptions	28
7	Hacking on HTTPretty	29
7.1	install development dependencies	29

7.2	next steps	29
8	License	31
9	Main contributors	33
10	Release Notes	35
10.1	Release 1.1.1	35
10.2	Release 1.1.0	35
10.3	Release 1.0.5	35
10.4	Release 1.0.4	36
10.5	Release 1.0.3	36
10.6	Release 1.0.0	36
10.7	Release 0.9.4	36
10.8	Release 0.8.4	37
10.9	Release 0.6.5	37
10.10	Release 0.6.2	37
10.11	Release 0.6.1	37
10.12	Release 0.5.14	38
10.13	Release 0.5.12	38
10.14	Release 0.5.11	38
10.15	Release 0.5.10	38
10.16	Release 0.5.9	38
10.17	Release 0.5.8	38
11	Indices and tables	39
	Python Module Index	41
	Index	43

HTTP Client mocking tool for Python created by [Gabriel Falcão](#) . It provides a full fake TCP socket module. Inspired by [FakeWeb](#)

Looking for the [Github Repository](#) ?

Python Support:

- **3.6**
- **3.7**
- **3.8**
- **3.9**

[Github](#)

WHAT IS HTTPRETTY ?

Once upon a time a python developer wanted to use a RESTful api, everything was fine but until the day he needed to test the code that hits the RESTful API: what if the API server is down? What if its content has changed ?

Don't worry, HTTPretty is here for you:

```
import logging
import requests
import httpretty

from sure import expect

logging.getLogger('httpretty.core').setLevel(logging.DEBUG)

@httpretty.activate(allow_net_connect=False)
def test_yipit_api_returning_deals():
    httpretty.register_uri(httpretty.GET, "http://api.yipit.com/v1/deals/",
                           body='[{"title": "Test Deal"}]',
                           content_type="application/json")

    response = requests.get('http://api.yipit.com/v1/deals/')

    expect(response.json()).to.equal([{"title": "Test Deal"}])
```

1.1 A more technical description

HTTPretty is a python library that swaps the modules `socket` and `ssl` with fake implementations that intercept HTTP requests at the level of a TCP connection.

It is inspired on Ruby's [FakeWeb](#).

If you come from the Ruby programming language this would probably sound familiar :smiley:

1.2 Installing

Installing httpretty is as easy as:

```
pip install httpretty
```


2.1 expecting a simple response body

```
import requests
import httpretty

def test_one():
    httpretty.enable(verbose=True, allow_net_connect=False) # enable HTTPretty so that
    ↪ it will monkey patch the socket module
    httpretty.register_uri(httpretty.GET, "http://yipit.com/",
                           body="Find the best daily deals")

    response = requests.get('http://yipit.com')

    assert response.text == "Find the best daily deals"

    httpretty.disable() # disable afterwards, so that you will have no problems in code
    ↪ that uses that socket module
    httpretty.reset() # reset HTTPretty state (clean up registered urls and request
    ↪ history)
```

2.2 making assertions in a callback that generates the response body

```
import requests
import json
import httpretty

@httpretty.activate
def test_with_callback_response():
    def request_callback(request, uri, response_headers):
        content_type = request.headers.get('Content-Type')
        assert request.body == '{"nothing": "here"}', 'unexpected body: {}'.format(request.
    ↪ body)
        assert content_type == 'application/json', 'expected application/json but received
    ↪ Content-Type: {}'.format(content_type)
        return [200, response_headers, json.dumps({"hello": "world"})]

    httpretty.register_uri(
```

(continues on next page)

(continued from previous page)

```
httpretty.POST, "https://httpretty.example.com/api",
body=request_callback)

response = requests.post('https://httpretty.example.com/api', headers={'Content-Type':
↪ 'application/json'}, data={'nothing': 'here'})

expect(response.json()).to.equal({"hello": "world"})
```

2.3 Link headers

Tests link headers by using the *adding_headers* parameter.

```
import requests
from sure import expect
import httpretty

@httpretty.activate
def test_link_response():
    first_url = "http://foo-api.com/data"
    second_url = "http://foo-api.com/data?page=2"
    link_str = "<%s>; rel='next'" % second_url

    httpretty.register_uri(
        httpretty.GET,
        first_url,
        body='{"success": true}',
        status=200,
        content_type="text/json",
        adding_headers={"Link": link_str},
    )
    httpretty.register_uri(
        httpretty.GET,
        second_url,
        body='{"success": false}',
        status=500,
        content_type="text/json",
    )
    # Performs a request to `first_url` followed by some testing
    response = requests.get(first_url)
    expect(response.json()).to.equal({"success": True})
    expect(response.status_code).to.equal(200)
    next_url = response.links["next"]["url"]
    expect(next_url).to.equal(second_url)

    # Follow the next URL and perform some testing.
    response2 = requests.get(next_url)
    expect(response2.json()).to.equal({"success": False})
    expect(response2.status_code).to.equal(500)
```

MOTIVATION

When building systems that access external resources such as RESTful webservices, XMLRPC or even simple HTTP requests, we stumble in the problem:

“I’m gonna need to mock all those requests”

It can be a bit of a hassle to use something like `mock.Mock` to stub the requests, this can work well for low-level unit tests but when writing functional or integration tests we should be able to allow the http calls to go through the TCP socket module.

HTTPretty [monkey patches](#) Python’s `socket` core module with a fake version of the module.

Because HTTPretty implements a fake the modules `socket` and `ssl` you can use write tests to code against any HTTP library that use those modules.

ACKNOWLEDGEMENTS

4.1 Caveats

4.1.1 `forcing_headers` + Content-Length

When using the `forcing_headers` option make sure to add the header `Content-Length` otherwise calls using `requests` will try to load the response endlessly.

4.1.2 Supported Libraries

Because HTTPretty works in the socket level it should work with any HTTP client libraries, although it is `battle tested` against:

- `requests`
- `httplib2`
- `urllib2`

API REFERENCE

5.1 register_uri

classmethod `httpretty.register_uri`(*method, uri, body*={'message': "HTTPretty :)"},
adding_headers=None, *forcing_headers*=None, *status*=200,
responses=None, *match_querystring*=False, *priority*=0, ****headers**)

```
import httpretty

def request_callback(request, uri, response_headers):
    content_type = request.headers.get('Content-Type')
    assert request.body == '{"nothing": "here"}', 'unexpected body: {}'.
    ↪format(request.body)
    assert content_type == 'application/json', 'expected application/json but
    ↪received Content-Type: {}'.format(content_type)
    return [200, response_headers, json.dumps({"hello": "world"})]

httpretty.register_uri(
    HTTPretty.POST, "https://httpretty.example.com/api",
    body=request_callback)

with httpretty.enabled():
    requests.post('https://httpretty.example.com/api', data='{"nothing": "here"}',
    ↪headers={'Content-Type': 'application/json'})

assert httpretty.latest_requests[-1].url == 'https://httpbin.org/ip'
```

Parameters

- **method** – one of `httpretty.GET`, `httpretty.PUT`, `httpretty.POST`, `httpretty.DELETE`, `httpretty.HEAD`, `httpretty.PATCH`, `httpretty.OPTIONS`, `httpretty.CONNECT`
- **uri** – a string or regex pattern (e.g.: “`https://httpbin.org/ip`”)
- **body** – a string, defaults to `{"message": "HTTPretty :)}"`
- **adding_headers** – dict - headers to be added to the response
- **forcing_headers** – dict - headers to be forcefully set in the response

- **status** – an integer, defaults to **200**
- **responses** – a list of entries, ideally each created with [Response\(\)](#)
- **priority** – an integer, useful for setting higher priority over previously registered urls. defaults to zero
- **match_querystring** – bool - whether to take the querystring into account when matching an URL
- **headers** – headers to be added to the response

Warning: When using a port in the request, add a trailing slash if no path is provided otherwise Httpretty will not catch the request. Ex: `httpretty.register_uri(httpretty.GET, 'http://fakeuri.com:8080/', body='{"hello": "world"}')`

5.2 enable

classmethod `httpretty.enable(allow_net_connect=True, verbose=False)`

Enables HTTPretty.

Parameters

- **allow_net_connect** – boolean to determine if unmatched requests are forwarded to a real network connection OR throw [httpretty.errors.UnmockedError](#).
- **verbose** – boolean to set HTTPretty’s logging level to DEBUG

```
import re, json
import httpretty

httpretty.enable(allow_net_connect=True, verbose=True)

httpretty.register_uri(
    httpretty.GET,
    re.compile(r'http://.*'),
    body=json.dumps({'man': 'in', 'the': 'middle'})
)

response = requests.get('https://foo.bar/foo/bar')

response.json().should.equal({
    "man": "in",
    "the": "middle",
})
```

Warning: after calling this method the original `socket` is replaced with [httpretty.core.fakesock](#). Make sure to call [disable\(\)](#) after done with your tests or use the `httpretty.enabled` as decorator or `context-manager`

5.3 disable

classmethod `httpretty.disable()`

Disables HTTPretty entirely, putting the original `socket` module back in its place.

```
import re, json
import httpretty

httpretty.enable()
# request passes through fake socket
response = requests.get('https://httpbin.org')

httpretty.disable()
# request uses real python socket module
response = requests.get('https://httpbin.org')
```

Note: This method does not call `httpretty.core.reset()` automatically.

5.4 is_enabled

classmethod `httpretty.is_enabled()`

Check if HTTPretty is enabled

Returns bool

```
import httpretty

httpretty.enable()
assert httpretty.is_enabled() == True

httpretty.disable()
assert httpretty.is_enabled() == False
```

5.5 last_request

`httpretty.last_request()`

Returns the last *HTTPrettyRequest*

5.6 latest_requests

`httpretty.latest_requests()`
returns the history of made requests

5.7 activate

`httpretty.activate`
alias of `httpretty.core.httprettified`

5.8 httprettified

`httpretty.core.httprettified(test=None, allow_net_connect=True, verbose=False)`
decorator for test functions

Tip: Also available under the alias `httpretty.activate()`

Parameters `test` – a callable

example usage with `nosetests`

```
import sure
from httpretty import httprettified

@httprettified
def test_using_nosetests():
    httpretty.register_uri(
        httpretty.GET,
        'https://httpbin.org/ip'
    )

    response = requests.get('https://httpbin.org/ip')

    response.json().should.equal({
        "message": "HTTPretty :)"
    })
```

example usage with `unittest` module

```
import unittest
from sure import expect
from httpretty import httprettified

@httprettified
class TestWithPyUnit(unittest.TestCase):
    def test_httpbin(self):
        httpretty.register_uri(httpretty.GET, 'https://httpbin.org/ip')
        response = requests.get('https://httpbin.org/ip')
```

(continues on next page)

(continued from previous page)

```
expect(response.json()).to.equal({
    "message": "HTTPretty :)"
})
```

5.9 enabled

`httpretty.enabled`

alias of `httpretty.core.httprettized`

5.10 httprettized

class `httpretty.core.httprettized`(*allow_net_connect=True, verbose=False*)
 context-manager for enabling HTTPretty.

Tip: Also available under the alias `httpretty.enabled()`

```
import json
import httpretty

httpretty.register_uri(httpretty.GET, 'https://httpbin.org/ip', body=json.dumps({
    ↪ 'origin': '42.42.42.42'}))
with httpretty.enabled():
    response = requests.get('https://httpbin.org/ip')

assert httpretty.latest_requests[-1].url == 'https://httpbin.org/ip'
assert response.json() == {'origin': '42.42.42.42'}
```

5.11 HTTPrettyRequest

class `httpretty.core.HTTPrettyRequest`(*headers, body="", sock=None, path_encoding='iso-8859-1'*)

Represents a HTTP request. It takes a valid multi-line, `\r\n` separated string with HTTP headers and parse them out using the internal `parse_request` method.

It also replaces the *rfile* and *wfile* attributes with `io.BytesIO` instances so that we guarantee that it won't make any I/O, neither for writing nor reading.

It has some convenience attributes:

`headers` -> a mimetype object that can be cast into a dictionary, contains all the request headers

`protocol` -> the protocol of this host, inferred from the port of the underlying fake TCP socket.

`host` -> the hostname of this request.

`url` -> the full url of this request.

`path` -> the path of the request.

`method` -> the HTTP method used in this request.

`querystring` -> a dictionary containing lists with the attributes. Please notice that if you need a single value from a query string you will need to get it manually like:

`body` -> the request body as a string.

`parsed_body` -> the request body parsed by `parse_request_body`.

```
>>> request.querystring
{'name': ['Gabriel Falcao']}
>>> print request.querystring['name'][0]
```

property method

the HTTP method used in this request

parse_querystring(*qs*)

parses an UTF-8 encoded query string into a dict of string lists

Parameters `qs` – a querystring

Returns a dict of lists

parse_request_body(*body*)

Attempt to parse the post based on the content-type passed. Return the regular body if not

Parameters `body` – string

Returns a python object such as dict or list in case the deserialization succeeded. Else returns the given param body

property protocol

the protocol used in this request

querystring

a dictionary containing parsed request body or None if HTTPrettyRequest doesn't know how to parse it. It currently supports parsing body data that was sent under the content-type headers values: ``application/json or application/x-www-form-urlencoded

property url

the full url of this recorded request

5.12 HTTPrettyRequestEmpty

class `httpretty.core.HTTPrettyRequestEmpty`

Represents an empty [HTTPrettyRequest](#) where all its properties are somehow empty or None

5.13 FakeSockFile

class `httpretty.core.FakeSockFile`

Fake socket file descriptor. Under the hood all data is written in a temporary file, giving it a real file descriptor number.

5.14 FakeSSLSocket

class httpretty.core.**FakeSSLSocket**(*sock*, **args*, ***kw*)
 Shorthand for [fakesock](#)

5.15 URIInfo

class httpretty.**URIInfo**(*username=""*, *password=""*, *hostname=""*, *port=80*, *path="/"*, *query=""*, *fragment=""*,
scheme="", *last_request=None*)
 Internal representation of [URIs](#)

Tip: all arguments are optional

Parameters

- **username** –
- **password** –
- **hostname** –
- **port** –
- **path** –
- **query** –
- **fragment** –
- **scheme** –
- **last_request** –

classmethod **from_uri**(*uri*, *entry*)

Parameters

- **uri** – string
- **entry** – an instance of [Entry](#)

full_url(*use_querystring=True*)

Parameters **use_querystring** – bool

Returns a string with the full url with the format {scheme}://{credentials}{domain}{path}{query}

get_full_domain()

Returns a string in the form {domain}:{port} or just the domain if the port is 80 or 443

5.16 URIMatcher

class httpretty.URIMatcher(*uri, entries, match_querystring=False, priority=0*)

get_next_entry(*method, info, request*)

Cycle through available responses, but only once. Any subsequent requests will receive the last response

5.17 Entry

class httpretty.Entry(*method, uri, body, adding_headers=None, forcing_headers=None, status=200, streaming=False, **headers*)

Created by [register_uri\(\)](#) and stored in memory as internal representation of a HTTP request/response definition.

Parameters

- **method** (*str*) – One of httpretty.GET, httpretty.PUT, httpretty.POST, httpretty.DELETE, httpretty.HEAD, httpretty.PATCH, httpretty.OPTIONS, httpretty.CONNECT.
- **uri** (*str/re.Pattern*) – The URL to match
- **adding_headers** (*dict*) – Extra headers to be added to the response
- **forcing_headers** (*dict*) – Overwrite response headers.
- **status** (*int*) – The status code for the response, defaults to 200.
- **streaming** (*bool*) – Whether should stream the response into chunks via generator.
- **headers** – Headers to inject in the faked response.

Returns containing the request-matching metadata.

Return type httpretty.Entry

Warning: When using the `forcing_headers` option make sure to add the header `Content-Length` to match at most the total body length, otherwise some HTTP clients can hang indefinitely.

fill_filekind(*fk*)

writes HTTP Response data to a file descriptor

Parm fk a file-like object

Warning: side-effect: this method moves the cursor of the given file object to zero

normalize_headers(*headers*)

Normalize keys in header names so that `Content-type` becomes `content-type`

Parameters headers – dict

Returns dict

validate()

validates the body size with the value of the `Content-Length` header

MODULES

6.1 Core

class httpretty.core.**EmptyRequestHeaders**

A dict subclass used as internal representation of empty request headers

class httpretty.core.**Entry**(*method, uri, body, adding_headers=None, forcing_headers=None, status=200, streaming=False, **headers*)

Created by [register_uri\(\)](#) and stored in memory as internal representation of a HTTP request/response definition.

Parameters

- **method** (*str*) – One of httpretty.GET, httpretty.PUT, httpretty.POST, httpretty.DELETE, httpretty.HEAD, httpretty.PATCH, httpretty.OPTIONS, httpretty.CONNECT.
- **uri** (*str/re.Pattern*) – The URL to match
- **adding_headers** (*dict*) – Extra headers to be added to the response
- **forcing_headers** (*dict*) – Overwrite response headers.
- **status** (*int*) – The status code for the response, defaults to 200.
- **streaming** (*bool*) – Whether should stream the response into chunks via generator.
- **headers** – Headers to inject in the faked response.

Returns containing the request-matching metadata.

Return type httpretty.Entry

Warning: When using the `forcing_headers` option make sure to add the header `Content-Length` to match at most the total body length, otherwise some HTTP clients can hang indefinitely.

fill_filekind(*fk*)

writes HTTP Response data to a file descriptor

Parm fk a file-like object

Warning: side-effect: this method moves the cursor of the given file object to zero

normalize_headers(*headers*)

Normalize keys in header names so that COntent-tYPe becomes content-type

Parameters *headers* – dict

Returns dict

validate()

validates the body size with the value of the Content-Length header

class httpretty.core.**FakeSSLSocket**(*sock, *args, **kw*)

Shorthand for *fakesock*

class httpretty.core.**FakeSockFile**

Fake socket file descriptor. Under the hood all data is written in a temporary file, giving it a real file descriptor number.

class httpretty.core.**HTTPrettyRequest**(*headers, body="", sock=None, path_encoding='iso-8859-1'*)

Represents a HTTP request. It takes a valid multi-line, \r\n separated string with HTTP headers and parse them out using the internal *parse_request* method.

It also replaces the *rfile* and *wfile* attributes with *io.BytesIO* instances so that we guarantee that it won't make any I/O, neither for writing nor reading.

It has some convenience attributes:

headers -> a mimetype object that can be cast into a dictionary, contains all the request headers

protocol -> the protocol of this host, inferred from the port of the underlying fake TCP socket.

host -> the hostname of this request.

url -> the full url of this request.

path -> the path of the request.

method -> the HTTP method used in this request.

querystring -> a dictionary containing lists with the attributes. Please notice that if you need a single value from a query string you will need to get it manually like:

body -> the request body as a string.

parsed_body -> the request body parsed by *parse_request_body*.

```
>>> request.querystring
{'name': ['Gabriel Falcao']}
>>> print request.querystring['name'][0]
```

property method

the HTTP method used in this request

parse_querystring(*qs*)

parses an UTF-8 encoded query string into a dict of string lists

Parameters *qs* – a querystring

Returns a dict of lists

parse_request_body(*body*)

Attempt to parse the post based on the content-type passed. Return the regular body if not

Parameters *body* – string

Returns a python object such as dict or list in case the deserialization succeeded. Else returns the given param body

property protocol

the protocol used in this request

querystring

a dictionary containing parsed request body or None if HTTPrettyRequest doesn't know how to parse it. It currently supports parsing body data that was sent under the `content-type` headers values: `application/json` or `application/x-www-form-urlencoded`

property url

the full url of this recorded request

class httpretty.core.HTTPrettyRequestEmpty

Represents an empty [HTTPrettyRequest](#) where all its properties are somehow empty or None

class httpretty.core.URIInfo(*username="", password="", hostname="", port=80, path='/', query="", fragment="", scheme="", last_request=None*)

Internal representation of [URIs](#)

Tip: all arguments are optional

Parameters

- **username** –
- **password** –
- **hostname** –
- **port** –
- **path** –
- **query** –
- **fragment** –
- **scheme** –
- **last_request** –

classmethod from_uri(*uri, entry*)

Parameters

- **uri** – string
- **entry** – an instance of [Entry](#)

full_url(*use_querystring=True*)

Parameters **use_querystring** – bool

Returns a string with the full url with the format `{scheme}://{credentials}{domain}{path}{query}`

get_full_domain()

Returns a string in the form `{domain}:{port}` or just the domain if the port is 80 or 443

`httpretty.core.create_fake_connection(address, timeout=<object object>, source_address=None)`
 drop-in replacement for `socket.create_connection()`

`httpretty.core.fake_getaddrinfo(host, port, family=None, socktype=None, proto=None, flags=None)`
 drop-in replacement for `socket.getaddrinfo()`

`httpretty.core.fake_gethostbyname(host)`
 drop-in replacement for `socket.gethostbyname()`

`httpretty.core.fake_gethostname()`
 drop-in replacement for `socket.gethostname()`

`httpretty.core.fake_wrap_socket(orig_wrap_socket_fn, *args, **kw)`
 drop-in replacement for `py:func:ssl.wrap_socket`

class `httpretty.core.fakesock`
 fake `socket`

class `socket(family=<AddressFamily.AF_INET: 2>, type=<SocketKind.SOCK_STREAM: 1>, proto=0, fileno=None)`

drop-in replacement for `socket.socket`

bind(*address*)

bind_truesock(*address*)

close()

connect(*address*)

connect_truesock(*request=None, address=None*)

create_socket(*address=None*)

fileno()

forward_and_trace(*function_name, *a, **kw*)

getpeercert(**a, **kw*)

property host

makefile(*mode='r', bufsize=-1*)

Returns this fake socket's own tempfile buffer.

If there is an entry associated with the socket, the file descriptor gets filled in with the entry data before being returned.

real_sendall(*data, *args, **kw*)

Sends data to the remote server. This method is called when HTTPretty identifies that someone is trying to send non-http data.

The received bytes are written in this socket's tempfile buffer so that HTTPretty can return it accordingly when necessary.

real_socket_is_connected()

recv(*bufferize=0, *args, **kwargs*)

recv_into(**args, **kwargs*)

recvfrom(**args, **kwargs*)

recvfrom_into(**args, **kwargs*)

send(*data, *args, **kwargs*)

```

sendall(data, *args, **kw)
sendto(*args, **kwargs)
setsockopt(level, optname, value)
settimeout(new_timeout)
ssl(sock, *args, **kw)

```

`httpretty.core.httprettified(test=None, allow_net_connect=True, verbose=False)`
decorator for test functions

Tip: Also available under the alias `httpretty.activate()`

Parameters `test` – a callable

example usage with `nosetests`

```

import sure
from httpretty import httprettified

@httprettified
def test_using_nosetests():
    httpretty.register_uri(
        httpretty.GET,
        'https://httpbin.org/ip'
    )

    response = requests.get('https://httpbin.org/ip')

    response.json().should.equal({
        "message": "HTTPretty :)"
    })

```

example usage with `unittest` module

```

import unittest
from sure import expect
from httpretty import httprettified

@httprettified
class TestWithPyUnit(unittest.TestCase):
    def test_httpbin(self):
        httpretty.register_uri(httpretty.GET, 'https://httpbin.org/ip')
        response = requests.get('https://httpbin.org/ip')
        expect(response.json()).to.equal({
            "message": "HTTPretty :)"
        })

```

`class httpretty.core.httprettized(allow_net_connect=True, verbose=False)`
`context-manager` for enabling HTTPretty.

Tip: Also available under the alias `httpretty.enabled()`

```
import json
import httpretty

httpretty.register_uri(httpretty.GET, 'https://httpbin.org/ip', body=json.dumps({
    ↪ 'origin': '42.42.42.42'}))
with httpretty.enabled():
    response = requests.get('https://httpbin.org/ip')

assert httpretty.latest_requests[-1].url == 'https://httpbin.org/ip'
assert response.json() == {'origin': '42.42.42.42'}
```

class httpretty.core.httpretty

manages HTTPretty’s internal request/response registry and request matching.

classmethod Response(*body, method=None, uri=None, adding_headers=None, forcing_headers=None, status=200, streaming=False, **kw*)

Shortcut to create an [Entry](#) that takes the body as first positional argument.

See also:

the parameters of this function match those of the [Entry](#) constructor.

Parameters

- **body** (*str*) – The body to return as response..
- **method** (*str*) – One of httpretty.GET, httpretty.PUT, httpretty.POST, httpretty.DELETE, httpretty.HEAD, httpretty.PATCH, httpretty.OPTIONS, httpretty.CONNECT.
- **uri** (*str/re.Pattern*) – The URL to match
- **adding_headers** (*dict*) – Extra headers to be added to the response
- **forcing_headers** (*dict*) – Overwrite **any** response headers, even “Content-Length”.
- **status** (*int*) – The status code for the response, defaults to 200.
- **streaming** (*bool*) – Whether should stream the response into chunks via generator.
- **kwargs** – Keyword-arguments are forwarded to [Entry](#)

Returns containing the request-matching metadata.

Return type httpretty.Entry

classmethod disable()

Disables HTTPretty entirely, putting the original [socket](#) module back in its place.

```
import re, json
import httpretty

httpretty.enable()
# request passes through fake socket
response = requests.get('https://httpbin.org')

httpretty.disable()
# request uses real python socket module
response = requests.get('https://httpbin.org')
```

Note: This method does not call `httpretty.core.reset()` automatically.

classmethod `enable(allow_net_connect=True, verbose=False)`

Enables HTTPretty.

Parameters

- **allow_net_connect** – boolean to determine if unmatched requests are forwarded to a real network connection OR throw `httpretty.errors.UnmockedError`.
- **verbose** – boolean to set HTTPretty’s logging level to DEBUG

```
import re, json
import httpretty

httpretty.enable(allow_net_connect=True, verbose=True)

httpretty.register_uri(
    httpretty.GET,
    re.compile(r'http://.*'),
    body=json.dumps({'man': 'in', 'the': 'middle'})
)

response = requests.get('https://foo.bar/foo/bar')

response.json().should.equal({
    "man": "in",
    "the": "middle",
})
```

Warning: after calling this method the original `socket` is replaced with `httpretty.core.fakesock`. Make sure to call `disable()` after done with your tests or use the `httpretty.enabled` as decorator or `context-manager`

classmethod `historify_request(headers, body="", sock=None)`

appends request to a list for later retrieval

```
import httpretty

httpretty.register_uri(httpretty.GET, 'https://httpbin.org/ip', body='')
with httpretty.enabled():
    requests.get('https://httpbin.org/ip')

assert httpretty.latest_requests[-1].url == 'https://httpbin.org/ip'
```

classmethod `is_enabled()`

Check if HTTPretty is enabled

Returns bool

```
import httpretty
```

(continues on next page)

(continued from previous page)

```
httpretty.enable()
assert httpretty.is_enabled() == True

httpretty.disable()
assert httpretty.is_enabled() == False
```

classmethod `match_http_address(hostname, port)`

Parameters

- **hostname** – a string
- **port** – an integer

Returns an URLMatcher or None

classmethod `match_https_hostname(hostname)`

Parameters **hostname** – a string

Returns an URLMatcher or None

classmethod `match_uriinfo(info)`

Parameters **info** – an *URIInfo*

Returns a 2-item tuple: (URLMatcher, *URIInfo*) or (None, [])

classmethod `playback(filename, allow_net_connect=True, verbose=False)`

```
import io
import json
import requests
import httpretty

with httpretty.record('/tmp/ip.json'):
    data = requests.get('https://httpbin.org/ip').json()

with io.open('/tmp/ip.json') as fd:
    assert data == json.load(fd)
```

Parameters **filename** – a string

Returns

a context-manager

classmethod `record(filename, indentation=4, encoding='utf-8', verbose=False, allow_net_connect=True, pool_manager_params=None)`

```
import io
import json
import requests
```

(continues on next page)

(continued from previous page)

```
import httpretty

with httpretty.record('/tmp/ip.json'):
    data = requests.get('https://httpbin.org/ip').json()

with io.open('/tmp/ip.json') as fd:
    assert data == json.load(fd)
```

Parameters

- **filename** – a string
- **indentation** – an integer, defaults to 4
- **encoding** – a string, defaults to “utf-8”

Returns

a context-manager

```
classmethod register_uri(method, uri, body='{"message": "HTTPretty :)"}', adding_headers=None,
                        forcing_headers=None, status=200, responses=None,
                        match_querystring=False, priority=0, **headers)
```

```
import httpretty

def request_callback(request, uri, response_headers):
    content_type = request.headers.get('Content-Type')
    assert request.body == '{"nothing": "here"}', 'unexpected body: {}'.
    ↪format(request.body)
    assert content_type == 'application/json', 'expected application/json but
    ↪received Content-Type: {}'.format(content_type)
    return [200, response_headers, json.dumps({"hello": "world"})]

httpretty.register_uri(
    HTTPretty.POST, "https://httpretty.example.com/api",
    body=request_callback)

with httpretty.enabled():
    requests.post('https://httpretty.example.com/api', data='{"nothing": "here"}
    ↪', headers={'Content-Type': 'application/json'})

assert httpretty.latest_requests[-1].url == 'https://httpbin.org/ip'
```

Parameters

- **method** – one of `httpretty.GET`, `httpretty.PUT`, `httpretty.POST`, `httpretty.DELETE`, `httpretty.HEAD`, `httpretty.PATCH`, `httpretty.OPTIONS`, `httpretty.CONNECT`
- **uri** – a string or regex pattern (e.g.: “https://httpbin.org/ip”)
- **body** – a string, defaults to `{"message": "HTTPretty :)}"`

- **adding_headers** – dict - headers to be added to the response
- **forcing_headers** – dict - headers to be forcefully set in the response
- **status** – an integer, defaults to **200**
- **responses** – a list of entries, ideally each created with [Response\(\)](#)
- **priority** – an integer, useful for setting higher priority over previously registered urls. defaults to zero
- **match_querystring** – bool - whether to take the querystring into account when matching an URL
- **headers** – headers to be added to the response

Warning: When using a port in the request, add a trailing slash if no path is provided otherwise Httpretty will not catch the request. Ex: `httpretty.register_uri(httpretty.GET, 'http://fakeuri.com:8080/', body='{"hello":"world"}')`

classmethod `reset()`

resets the internal state of HTTPretty, unregistering all URLs

`httpretty.core.url_fix(s, charset=None)`

escapes special characters

6.2 Http

`httpretty.http.last_requestline(sent_data)`

Find the last line in `sent_data` that can be parsed with `parse_requestline`

`httpretty.http.parse_requestline(s)`

`http://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html#sec5`

```
>>> parse_requestline('GET / HTTP/1.0')
('GET', '/', '1.0')
>>> parse_requestline('post /testurl htTP/1.1')
('POST', '/testurl', '1.1')
>>> parse_requestline('Im not a RequestLine')
Traceback (most recent call last):
...
ValueError: Not a Request-Line
```

6.3 Utils

6.4 Exceptions

exception `httpretty.errors.HTTPrettyError`

exception `httpretty.errors.UnmockedError` (*message='Failed to handle network request', request=None, address=None*)

HACKING ON HTTPRETTY

7.1 install development dependencies

Note: HTTPretty uses [GNU Make](#) as default build tool.

```
make dependencies
```

7.2 next steps

1. run the tests with make:

```
make tests
```

2. hack at will
3. commit, push etc
4. send a pull request

LICENSE

```
<HTTPretty - HTTP client mock for Python>  
Copyright (C) <2011-2020> Gabriel Falcão <gabriel@nacaolive.org>
```

Permission **is** hereby granted, free of charge, to **any** person obtaining a copy of this software **and** associated documentation files (the "**Software**"), to deal **in** the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, **and/or** sell copies of the Software, **and** to permit persons to whom the Software **is** furnished to do so, subject to the following conditions:

The above copyright notice **and** this permission notice shall be included **in all** copies **or** substantial portions of the Software.

THE SOFTWARE IS PROVIDED "**AS IS**", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

MAIN CONTRIBUTORS

HTTPretty has received [many contributions](#) but some folks made remarkable contributions and deserve extra credit:

- Andrew Gross ~> [@andrewgross](#)
- Hugh Saunders ~> [@hughsaunders](#)
- James Rowe ~> [@JNRowe](#)
- Matt Luongo ~> [@mhlungo](#)
- Steve Pulec ~> [@spulec](#)

RELEASE NOTES

10.1 Release 1.1.1

- Bugfix: `httpretty.disable()` injects `pyopenssl` into `urllib3` even if it originally wasn't #417
- Bugfix: "Incompatibility with boto3 S3 `put_object`" #416
- Bugfix: "Regular expression for URL -> `TypeError: wrap_socket() missing 1 required`" #423
- Bugfix: "Making requests to non-standard port throws `TimeoutError`" #387

10.2 Release 1.1.0

- Feature: Display mismatched URL within `UnmockedError` whenever possible. #388
- Feature: Display mismatched URL via logging. #419
- Add new properties to `httpretty.core.HTTPrettyRequest` (`protocol`, `host`, `url`, `path`, `method`).

Example usage:

```
import httpretty
import requests

@httpretty.activate(verbose=True, allow_net_connect=False)
def test_mismatches():
    requests.get('http://sql-server.local')
    requests.get('https://redis.local')
```

10.3 Release 1.0.5

- Bugfix: Support `socket.socketpair()` . #402
- Bugfix: Prevent exceptions from re-applying monkey patches. #406

10.4 Release 1.0.4

- Python 3.8 and 3.9 support. [#407](#)

10.5 Release 1.0.3

- Fix compatibility with urllib3>=1.26. [#410](#)

10.6 Release 1.0.0

- Drop Python 2 support.
- Fix usage with redis and improve overall real-socket passthrough. [#271](#).
- Fix TypeError: wrap_socket() missing 1 required positional argument: 'sock' ([#393](#))
- Merge pull request [#364](#)
- Merge pull request [#371](#)
- Merge pull request [#379](#)
- Merge pull request [#386](#)
- Merge pull request [#302](#)
- Merge pull request [#373](#)
- Merge pull request [#383](#)
- Merge pull request [#385](#)
- Merge pull request [#389](#)
- Merge pull request [#391](#)
- Fix simple typo: neighter -> neither.
- Updated documentation for register_uri concerning using ports.
- Clarify relation between enabled and httprettized in API docs.
- Align signature with builtin socket.

10.7 Release 0.9.4

Improvements:

- Official Python 3.6 support
- Normalized coding style to conform with PEP8 (partially)
- Add more API reference coverage in docstrings of members such as [httpretty.core.Entry](#)
- Continuous Integration building python 2.7 and 3.6
- Migrate from [pip](#) to [pipenv](#)

10.8 Release 0.8.4

Improvements:

- Refactored `core.py` and increased its unit test coverage to 80%. HTTPretty is slightly more robust now.

Bug fixes:

- POST requests being called twice [#100](#)

10.9 Release 0.6.5

Applied pull requests:

- continue on EAGAIN socket errors: [#102](#) by [kouk](#).
- Fix `fake_gethostbyname` for requests 2.0: [#101](#) by [mgood](#)
- Add a way to match the querystrings: [#98](#) by [ametaireau](#)
- Use common string case for URIInfo hostname comparison: [#95](#) by [mikewaters](#)
- Expose `httpretty.reset()` to public API: [#91](#) by [imankulov](#)
- Don't duplicate http ports number: [#89](#) by [mardiros](#)
- Adding `parsed_body` parameter to simplify checks: [#88](#) by [toumorokoshi](#)
- Use the real socket if it's not HTTP: [#87](#) by [mardiros](#)

10.10 Release 0.6.2

- Fixing bug of lack of trailing slashes [#73](#)
- Applied pull requests [#71](#) and [#72](#) by [@andresriancho](#)
- Keyword arg coercion fix by [@dupuy](#)
- [@papaeye](#) fixed content-length calculation.

10.11 Release 0.6.1

- New API, no more camel case and everything is available through a simple import:

```
import httpretty

@httpretty.activate
def test_function():
    # httpretty.register_uri(...)
    # make request...
    pass
```

- Re-organized module into submodules

10.12 Release 0.5.14

- Delegate calls to other methods on socket
- Normalized header strings
- Callbacks are more intelligent now
- Normalize urls matching for url quoting

10.13 Release 0.5.12

- HTTPretty doesn't hang when using other application protocols under a @httprettified decorated test.

10.14 Release 0.5.11

- Ability to know whether HTTPretty is or not enabled through `httpretty.is_enabled()`

10.15 Release 0.5.10

- Support to multiple methods per registered URL. Thanks @hughsaunders

10.16 Release 0.5.9

- Fixed python 3 support. Thanks @spulec

10.17 Release 0.5.8

- Support to *register regular expressions to match urls*
- *Body callback* support
- Python 3 support

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

h

- `httpretty`, [14](#)
- `httpretty.core`, [19](#)
- `httpretty.errors`, [28](#)
- `httpretty.http`, [28](#)
- `httpretty.utils`, [28](#)

B

`bind()` (*httpretty.core.fakesock.socket method*), 22
`bind_truesock()` (*httpretty.core.fakesock.socket method*), 22

C

`close()` (*httpretty.core.fakesock.socket method*), 22
`connect()` (*httpretty.core.fakesock.socket method*), 22
`connect_truesock()` (*httpretty.core.fakesock.socket method*), 22
`create_fake_connection()` (*in module httpretty.core*), 21
`create_socket()` (*httpretty.core.fakesock.socket method*), 22

D

`disable()` (*httpretty.core.httpretty class method*), 24

E

`EmptyRequestHeaders` (*class in httpretty.core*), 19
`enable()` (*httpretty.core.httpretty class method*), 25
`Entry` (*class in httpretty.core*), 19

F

`fake_getaddrinfo()` (*in module httpretty.core*), 22
`fake_gethostbyname()` (*in module httpretty.core*), 22
`fake_gethostname()` (*in module httpretty.core*), 22
`fake_wrap_socket()` (*in module httpretty.core*), 22
`fakesock` (*class in httpretty.core*), 22
`fakesock.socket` (*class in httpretty.core*), 22
`FakeSockFile` (*class in httpretty.core*), 20
`FakeSSLSocket` (*class in httpretty.core*), 20
`fileno()` (*httpretty.core.fakesock.socket method*), 22
`fill_filekind()` (*httpretty.core.Entry method*), 19
`forward_and_trace()` (*httpretty.core.fakesock.socket method*), 22
`from_uri()` (*httpretty.core.URIInfo class method*), 21
`full_url()` (*httpretty.core.URIInfo method*), 21

G

`get_full_domain()` (*httpretty.core.URIInfo method*), 21

`getpeercert()` (*httpretty.core.fakesock.socket method*), 22

H

`historify_request()` (*httpretty.core.httpretty class method*), 25
`host` (*httpretty.core.fakesock.socket property*), 22
`httprettified()` (*in module httpretty.core*), 23
`httprettized` (*class in httpretty.core*), 23
`httpretty`
 module, 14
`httpretty` (*class in httpretty.core*), 24
`httpretty.core`
 module, 19
`httpretty.errors`
 module, 28
`httpretty.http`
 module, 28
`httpretty.utils`
 module, 28
`HTTPrettyError`, 28
`HTTPrettyRequest` (*class in httpretty.core*), 20
`HTTPrettyRequestEmpty` (*class in httpretty.core*), 21

I

`is_enabled()` (*httpretty.core.httpretty class method*), 25

L

`last_requestline()` (*in module httpretty.http*), 28

M

`makefile()` (*httpretty.core.fakesock.socket method*), 22
`match_http_address()` (*httpretty.core.httpretty class method*), 26
`match_https_hostname()` (*httpretty.core.httpretty class method*), 26
`match_uriinfo()` (*httpretty.core.httpretty class method*), 26
`method` (*httpretty.core.HTTPrettyRequest property*), 20
module
 httpretty, 14
 httpretty.core, 19

`httpretty.errors`, 28
`httpretty.http`, 28
`httpretty.utils`, 28

N

`normalize_headers()` (*httpretty.core.Entry* method), 19

P

`parse_querystring()`
(*httpretty.core.HTTPrettyRequest* method), 20

`parse_request_body()`
(*httpretty.core.HTTPrettyRequest* method), 20

`parse_requestline()` (*in module httpretty.http*), 28

`playback()` (*httpretty.core.httpretty* class method), 26

`protocol` (*httpretty.core.HTTPrettyRequest* property), 21

Q

`querystring` (*httpretty.core.HTTPrettyRequest* attribute), 21

R

`real_sendall()` (*httpretty.core.fakesock.socket* method), 22

`real_socket_is_connected()`
(*httpretty.core.fakesock.socket* method), 22

`record()` (*httpretty.core.httpretty* class method), 26

`recv()` (*httpretty.core.fakesock.socket* method), 22

`recv_into()` (*httpretty.core.fakesock.socket* method), 22

`recvfrom()` (*httpretty.core.fakesock.socket* method), 22

`recvfrom_into()` (*httpretty.core.fakesock.socket* method), 22

`register_uri()` (*httpretty.core.httpretty* class method), 27

`reset()` (*httpretty.core.httpretty* class method), 28

`Response()` (*httpretty.core.httpretty* class method), 24

S

`send()` (*httpretty.core.fakesock.socket* method), 22

`sendall()` (*httpretty.core.fakesock.socket* method), 22

`sendto()` (*httpretty.core.fakesock.socket* method), 23

`setsockopt()` (*httpretty.core.fakesock.socket* method), 23

`settimeout()` (*httpretty.core.fakesock.socket* method), 23

`ssl()` (*httpretty.core.fakesock.socket* method), 23

U

`UnmockedError`, 28

`URIInfo` (*class in httpretty.core*), 21

`url` (*httpretty.core.HTTPrettyRequest* property), 21

`url_fix()` (*in module httpretty.core*), 28

V

`validate()` (*httpretty.core.Entry* method), 20